Open Source Programming Digital Assessment-1

Name: Tanmay Ahuja

RegNo. 18BIT0208

Slot: B2+TB2

Prof. Jayakumar Sadhasivam

GitHub Digital Assessment 1 JayakumarClassroom link:

https://github.com/JayakumarClassroom/fs20-ite1008-da1-b2-tanmay140

GitHub Digital Assessment 1 Repository link:

https://github.com/tanmay140/Digital-Assignmemt-1

GitHub Portfolio Website Repository Link:

https://github.com/tanmay140/tanmay140.github.io

Portfolio Website Direct Link:

https://tanmay140.github.io/

Topic: Case Study on GitHub Version Control

- **Step by step process of GitHub working methodology and different ways to access GitHub:**

1) Introduction to GitHub:

GitHub is a highly used software that is typically used for version control. It is helpful when more than just one person

is working on a project. Say for example, a software developer team wants to build a website and everyone has to update their codes simultaneously while working on the project. In this case, GitHub helps them to build a centralized repository where everyone can upload, edit, and manage the code files.

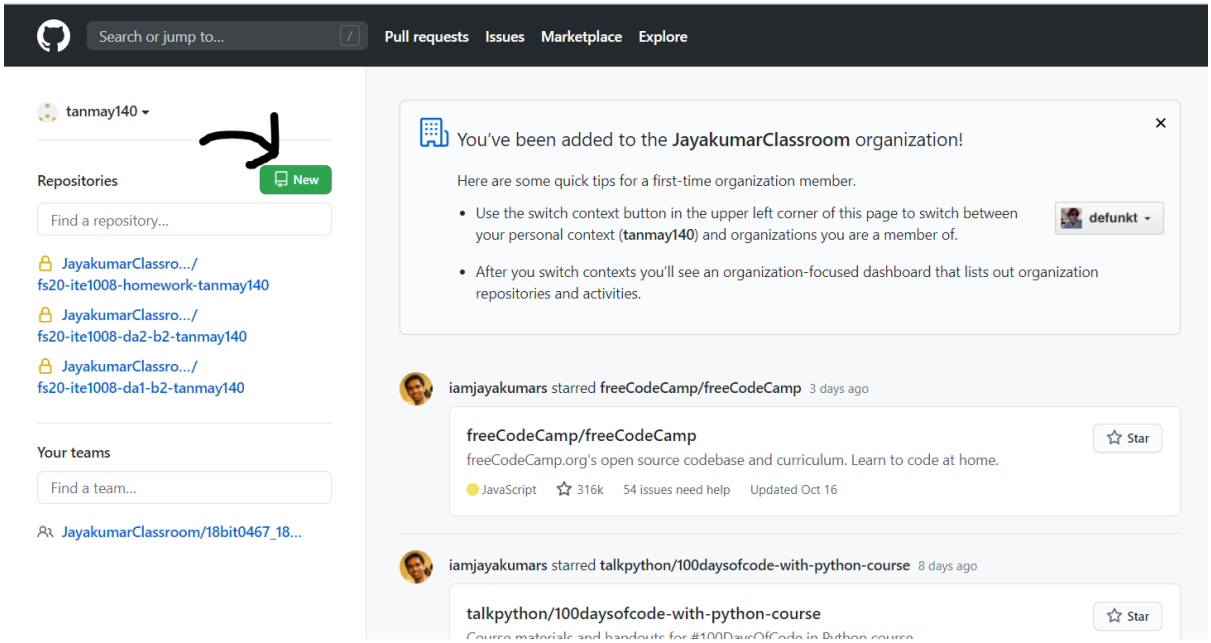
Some features of GitHub are:

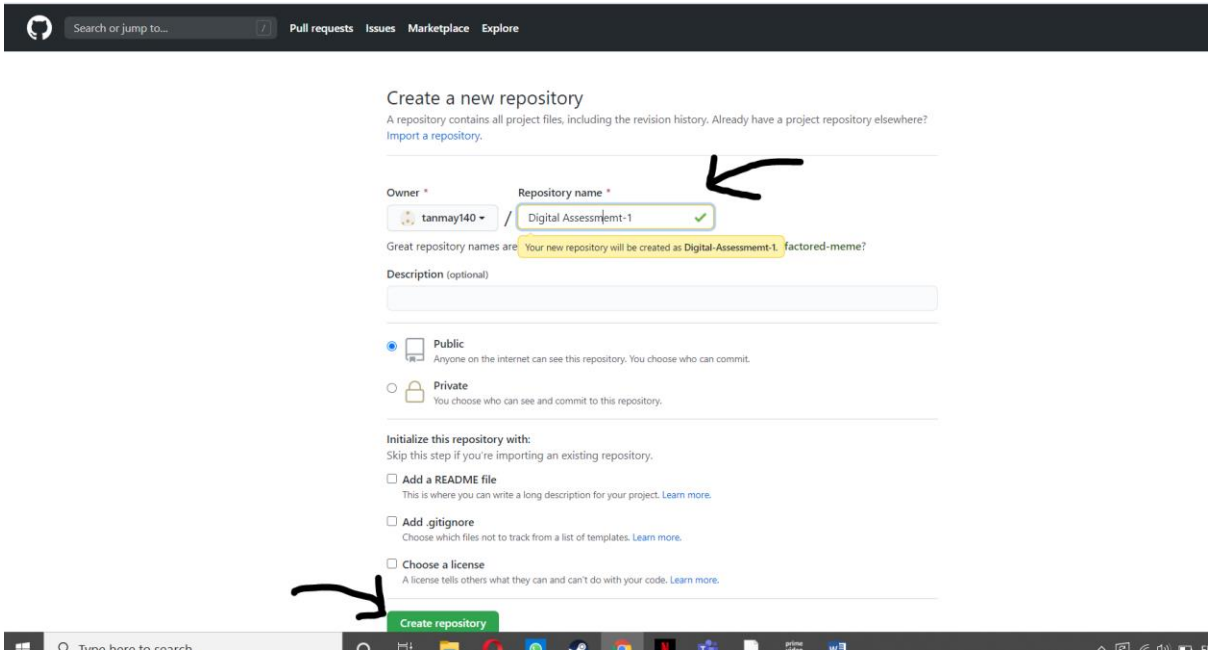
- ✓ GitHub provides you a beautiful visual interface which helps you to track or manage your version controlled projects locally.
- ✓ Once you register on GitHub, you can connect with social network and build a strong profile.

2) Creating a GitHub Repository:

A repository is a storage space where your project lives. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, images or any kind of a file in a repository. You need a GitHub repository when you have done some changes and are ready to be uploaded. This GitHub repository acts as your remote repository. So let me make your task easy, just follow these simple steps to create a GitHub repository:

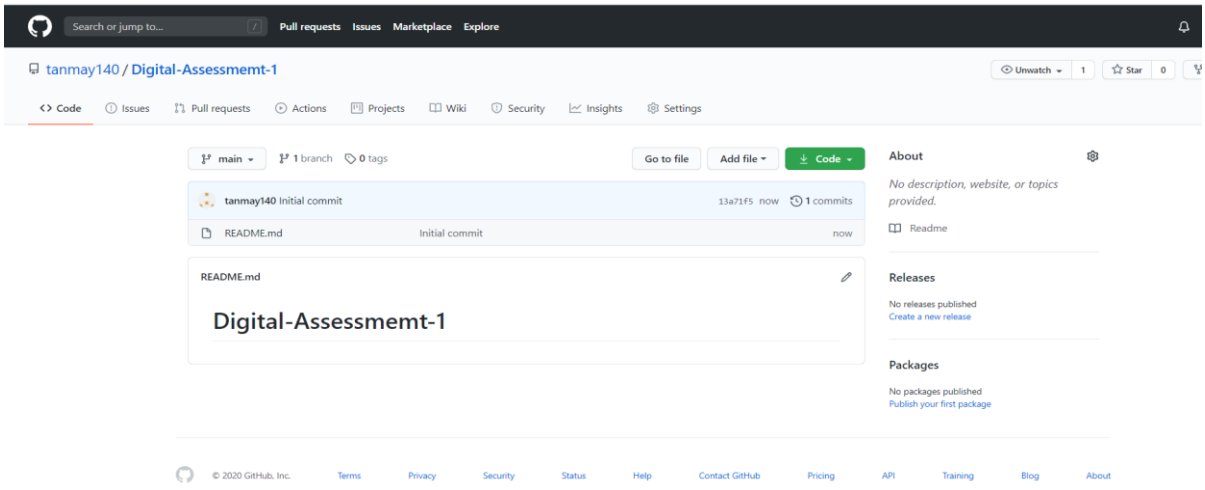
- ✓ Go to the link: https://github.com/ . Fill the sign up form and click on “Sign up for Github”.
- ✓ Click on “Start a new project”.





Public GitHub repository means that anyone can view the contents of this repository whereas in a private repository,

you can choose who can view the content. Also, private repository is a paid version. Also, if you refer the above screenshot, initialize the repository with a README file. This file contains the description of the file and once you check this box, this will be the first file inside your repository.



So our Repository is created successfully.

3) Create Branches and Perform Operations:

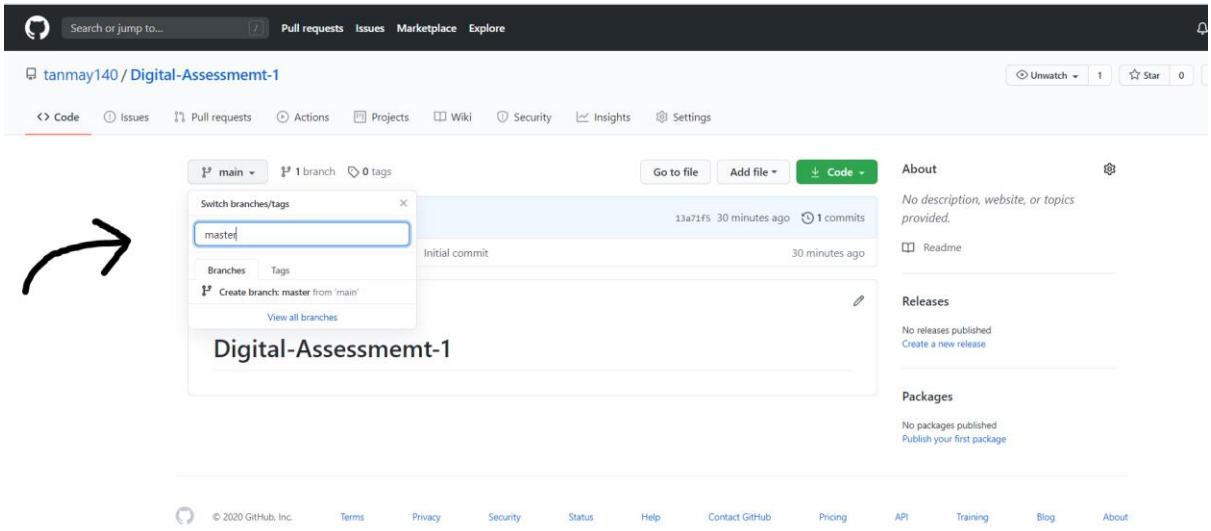
Branches help you to work on different versions of a repository at one time. Let's say you want to add a new feature (which is in the development phase), and you are afraid at the same time whether to make changes to your main project or not. This is where git branching comes to rescue. Branches allow you to move back and forth between the different states/versions of a project. In the above scenario, you can create a new branch and test the new feature without affecting the main branch. Once you are done with it, you can merge the changes from new branch to the main branch. Here the main branch is the master branch, which is there in your

repository by default. Refer to the below image for better understanding:

To create a branch in GitHub, follow the below steps:

✓ Click on the dropdown “Branch: master”

✓ As soon as you click on the branch, you can find an existing branch or you can create a new one. In my case, I am creating a new branch with a name “readme- changes”. Refer to the below screenshot for better understanding.



Once you have created a new branch, you have two branches in your repository now i.e. read-me (master branch) and readme-changes. The new branch is just the copy of master branch.

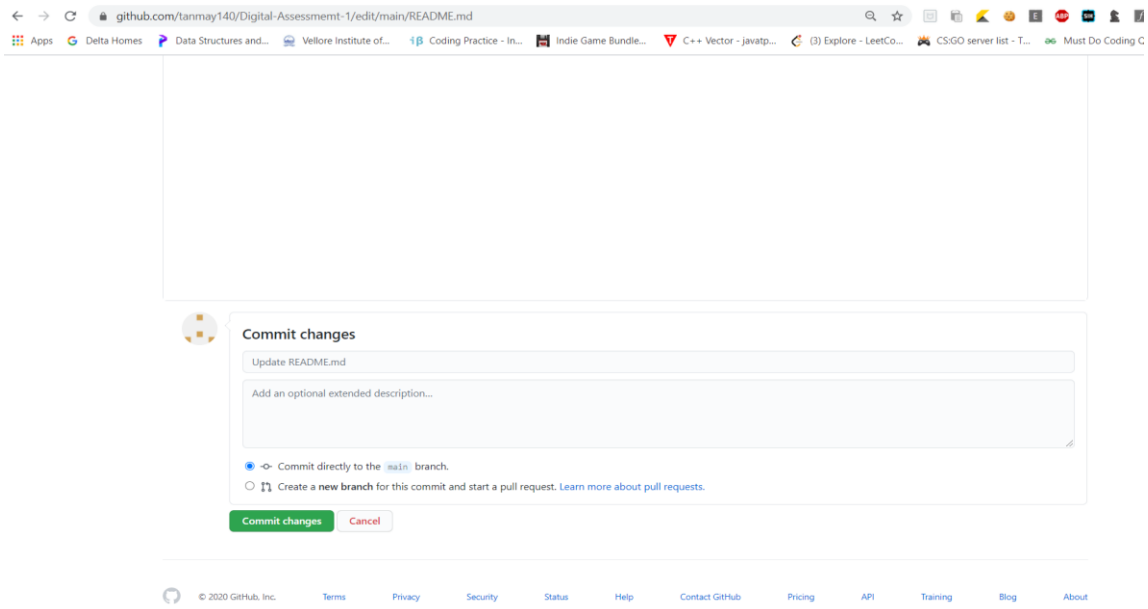
GitHub: Operations:

i) Commit Command:

This operation helps you to save the changes in your file. When you commit a file, you should always provide the

message, just to keep in the mind the changes done by you. Though this message is not compulsory but it is always recommended so that it can differentiate the various versions or commits you have done so far to your repository. These commit messages maintain the history of changes which in turn help other contributors to understand the file better. Now let's make our first commit, follow the below steps:

- Click on "readme- changes" file which we have just created.
- Click on the "edit" or a pencil icon in the righmost corner of the file.
- Once you click on that, an editor will open where you can type in the changes or anything.
- Write a commit message which identifies your changes.
- Click commit changes in the end.

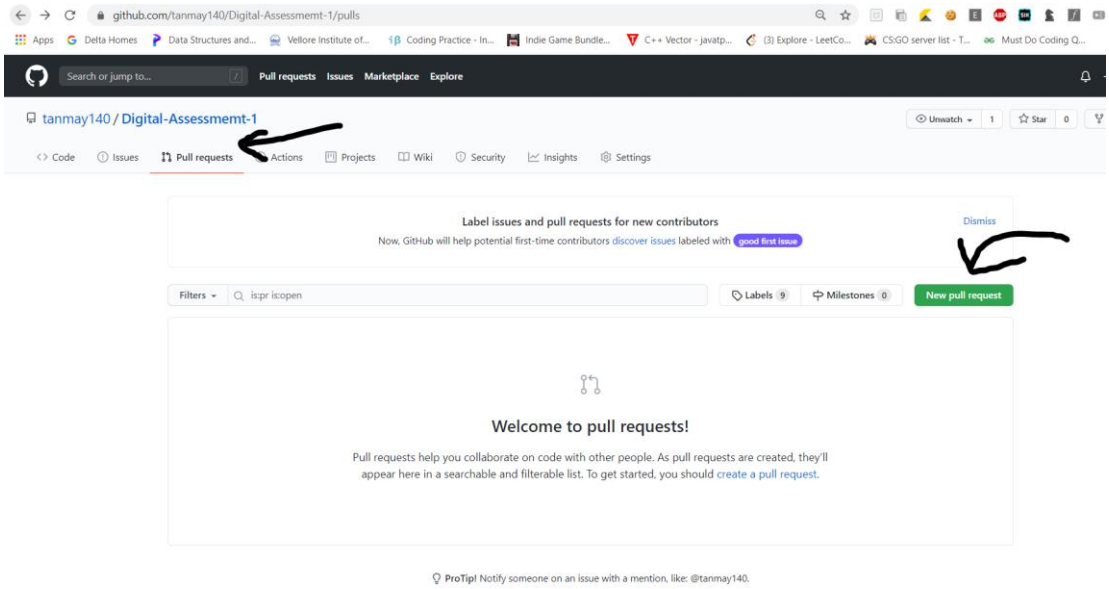


We have successfully made our first commit.

ii) <u>Pull Command:</u>

Pull command is the most important command in GitHub. It tells the changes done in the file and request other contributors to view it as well as merge it with the master branch. Once the commit is done, anyone can pull the file and can start a discussion over it. Once it's all done, you can merge the file. Pull command compares the changes which are done in the file and if there are any conflicts, you can manually resolve it. Now let us see different steps involved to pull request in GitHub.

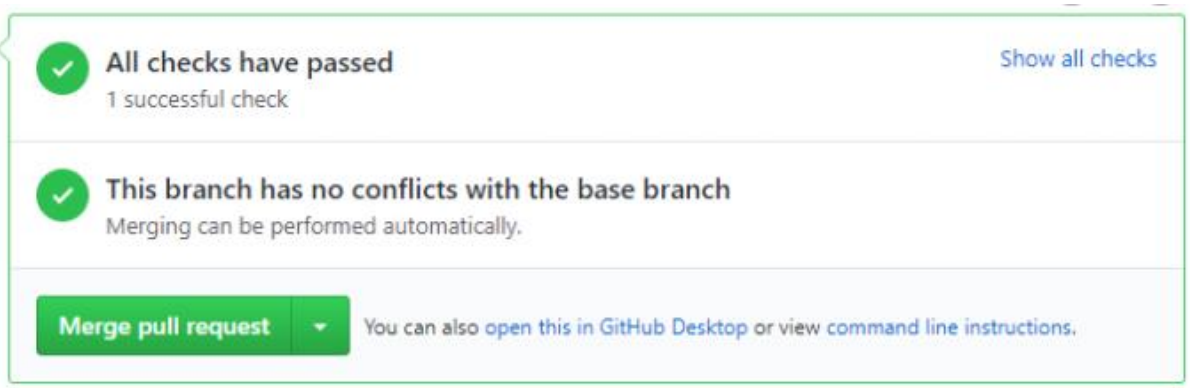
- Click the 'Pull requests' tab.
- Click 'New pull request'.
- Once you click on pull request, select the branch and click 'readme- changes' file to view changes between the two files present in our repository.
- Click "Create pull request".
- Enter any title, description to your changes and click on "Create pull request". Refer to the below screenshots.



iii) Merge Command:

Here comes the last command which merge the changes into the main master branch. We saw the changes in pink and green color, now let's merge the "readme- changes" file with the master branch/ read-me. Go through the below steps to merge pull request.

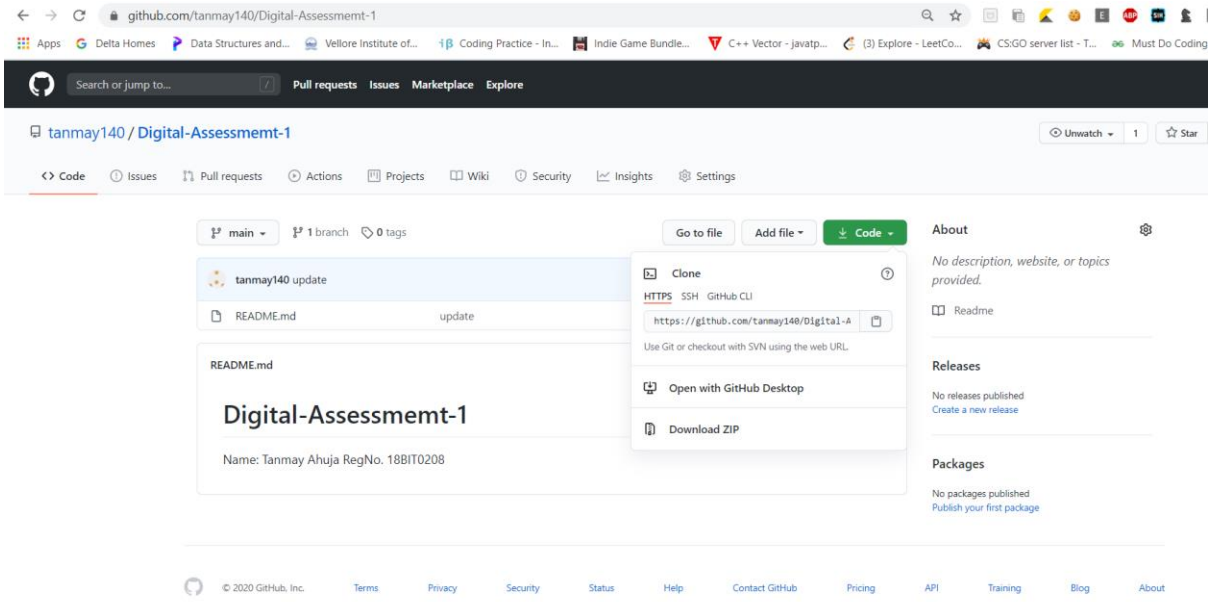
- Click on "Merge pull request" to merge the changes into master branch.
- Click "Confirm merge".
- You can delete the branch once all the changes have been incorporated and if there are no conflicts. Refer to the below screenshots.



4) Cloning and Forking GitHub Repository:

Cloning: Before I actually talk about cloning a GitHub repository, first let us understand why do we need to clone a repository. The answer is simple! Suppose you want to use

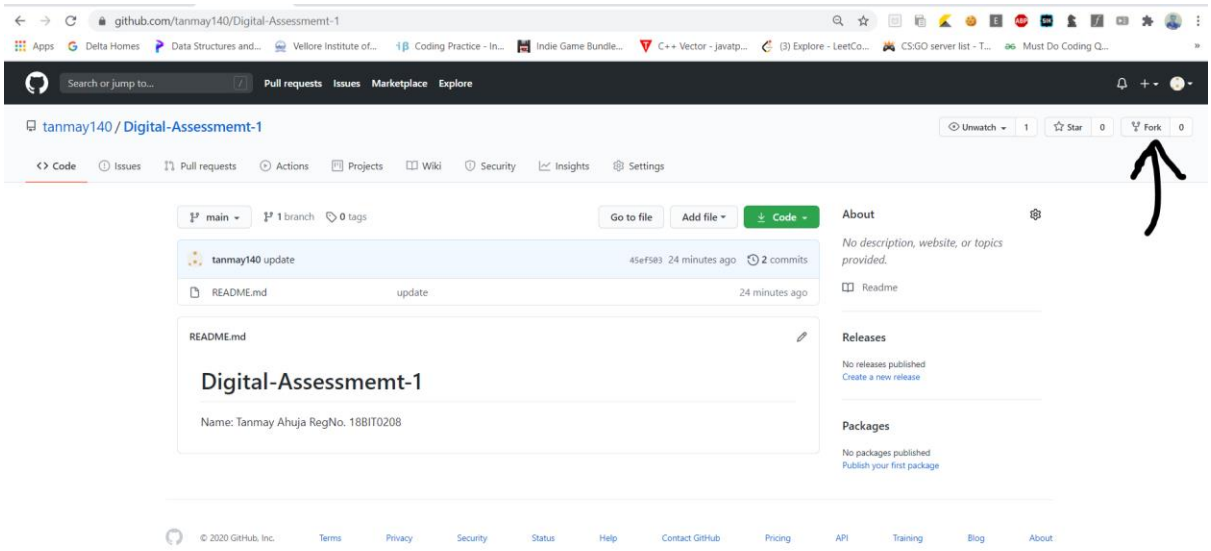
some code which is present in a public repository, you can directly copy the contents by cloning or downloading. Refer to the below screenshot for a better understanding.



Forking: First, let us talk about why do we need forking. Suppose, you need some code which is present in a public repository, under your repository and GitHub account. For this, we need to fork a repository.

Before we get started with forking, there are some important points which you should always keep in mind.

- Changes done to the original repository will be reflected back to the forked repository.
- If you make a change in forked repository, it will not be reflected to the original repository until and unless you have made a pull request.

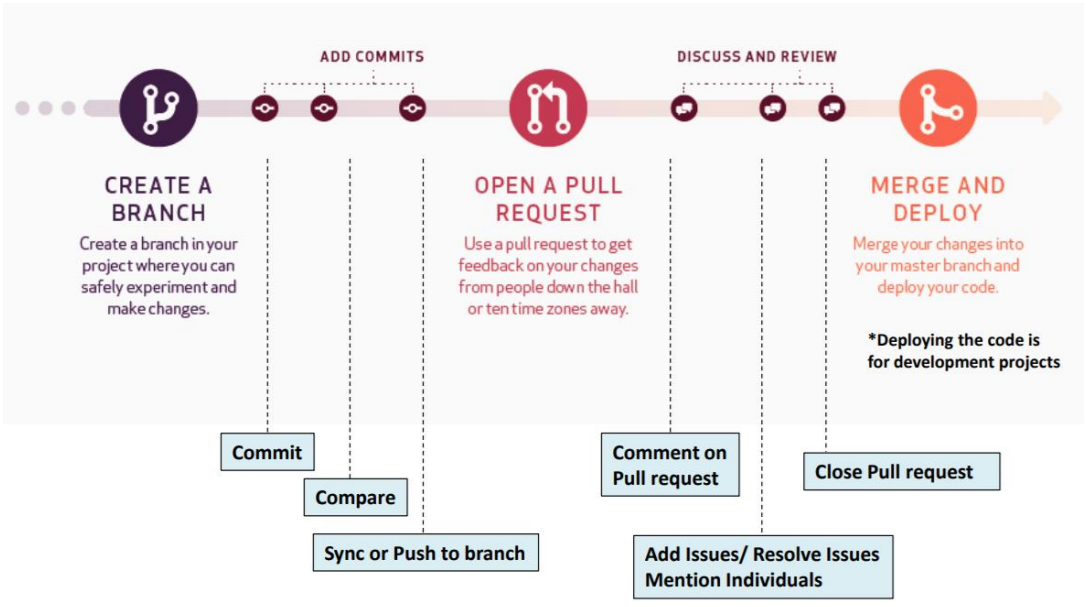


As soon as you click on “Fork”, it will take some time to fork the repository. Once done you will notice that the repository name is under your account.

5) <u>Watch and Star:</u>

Watch notifies us of all conversations over and above your @mentions, commits, comments on discussion. Star will favorite it but not show on your dashboards like watch

<u>OVERAL WORKFLOW</u>



- **Host your Personal Portfolio in GitHub and provide the screenshot of the project and version history**

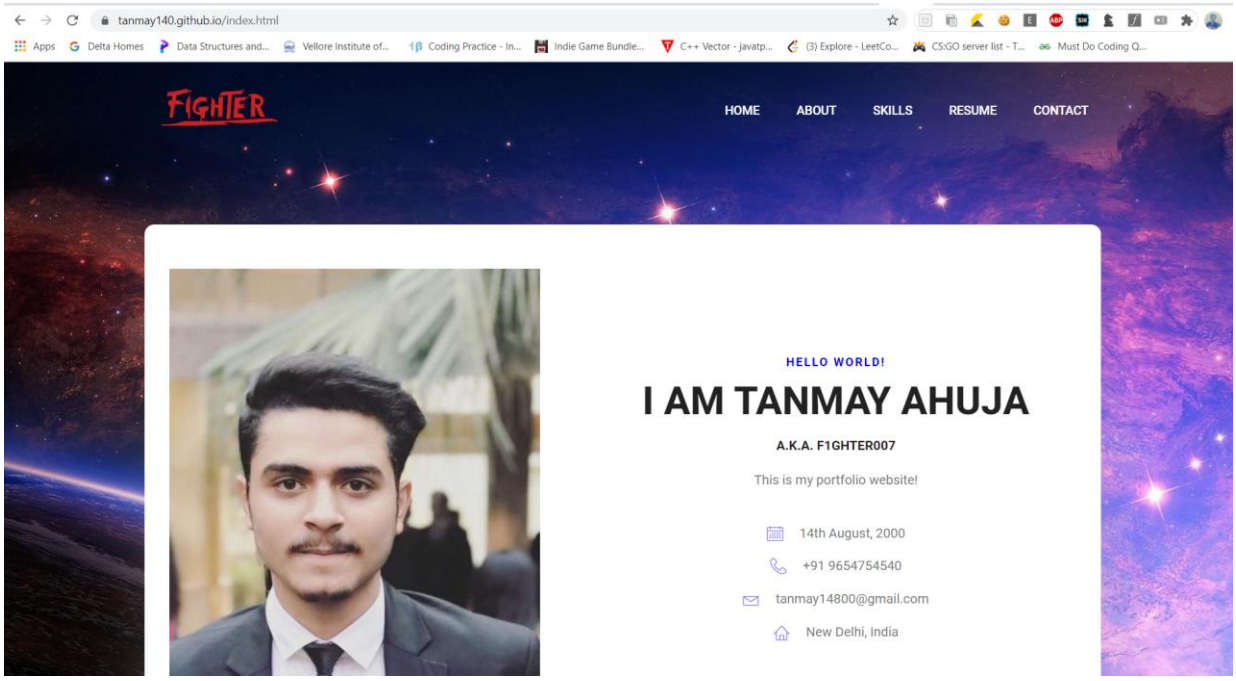
I have successfully hosted my Portfolio website on GitHub using GitHub pages. **The link to the website is:** https://tanmay140.github.io/

I have uploaded contents to my GitHub account in the Repository named as **tanmay140.github.io :**

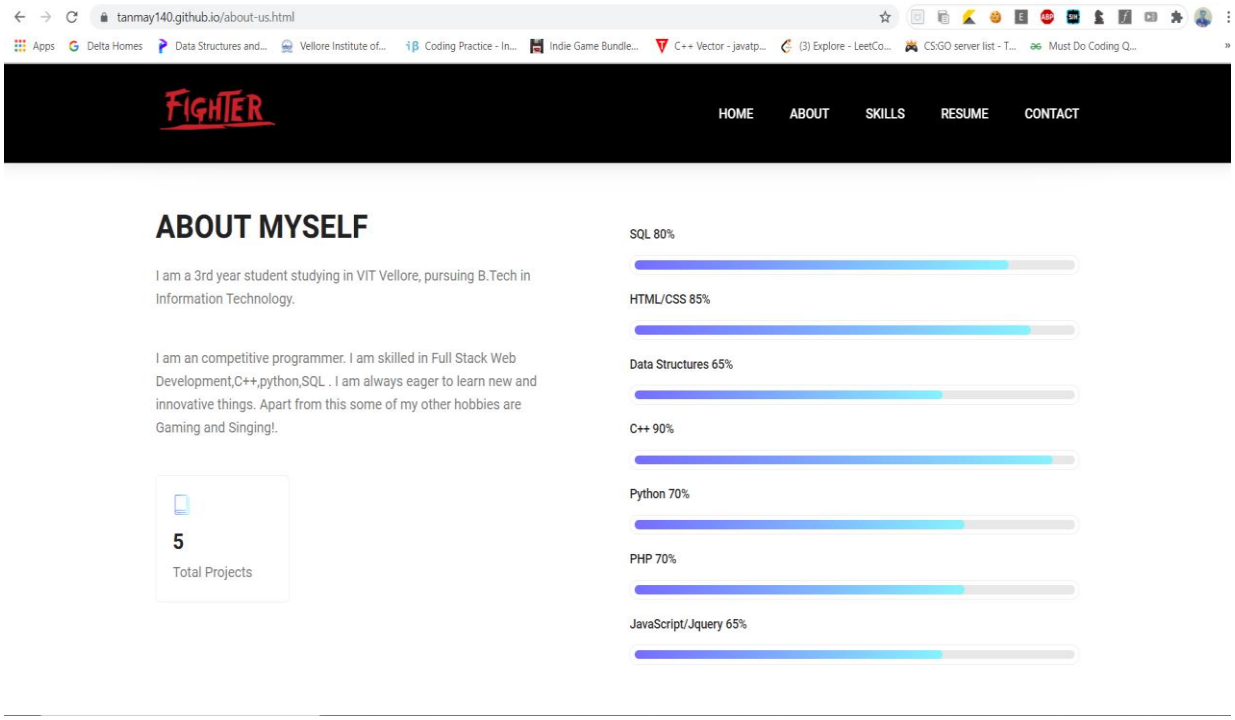
https://github.com/tanmay140/tanmay140.github.io

Screenshots of my Portfolio Website:

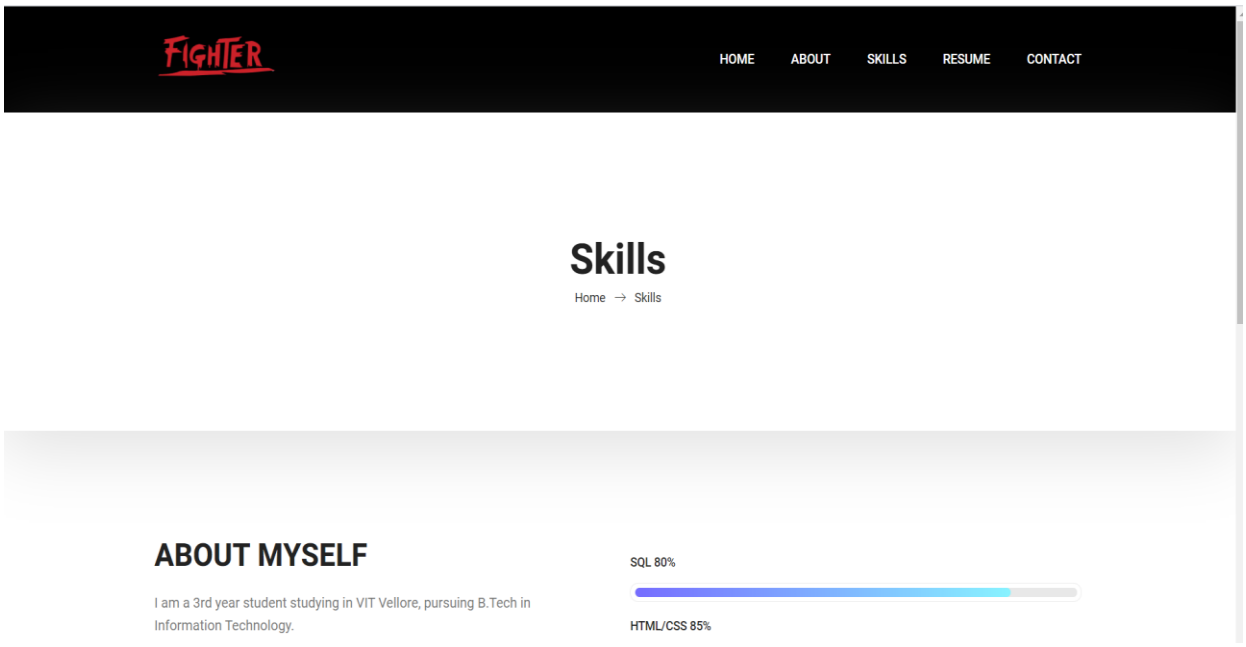
Home:



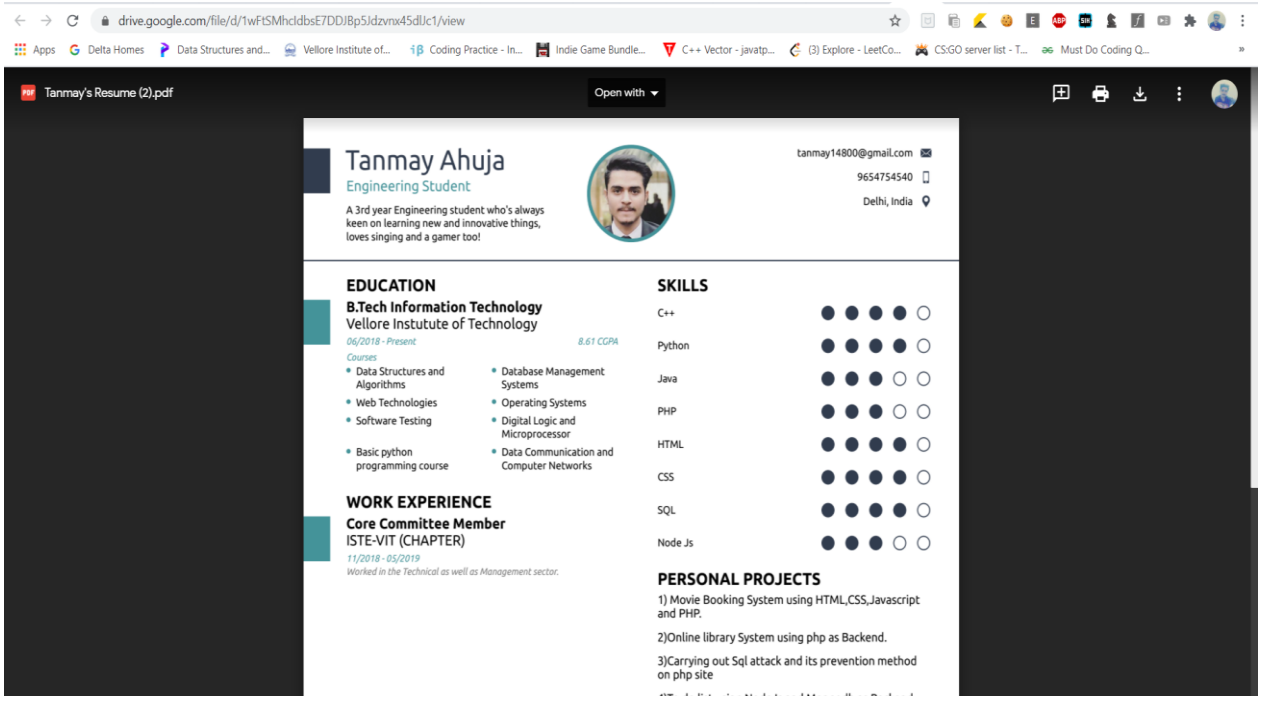
About:



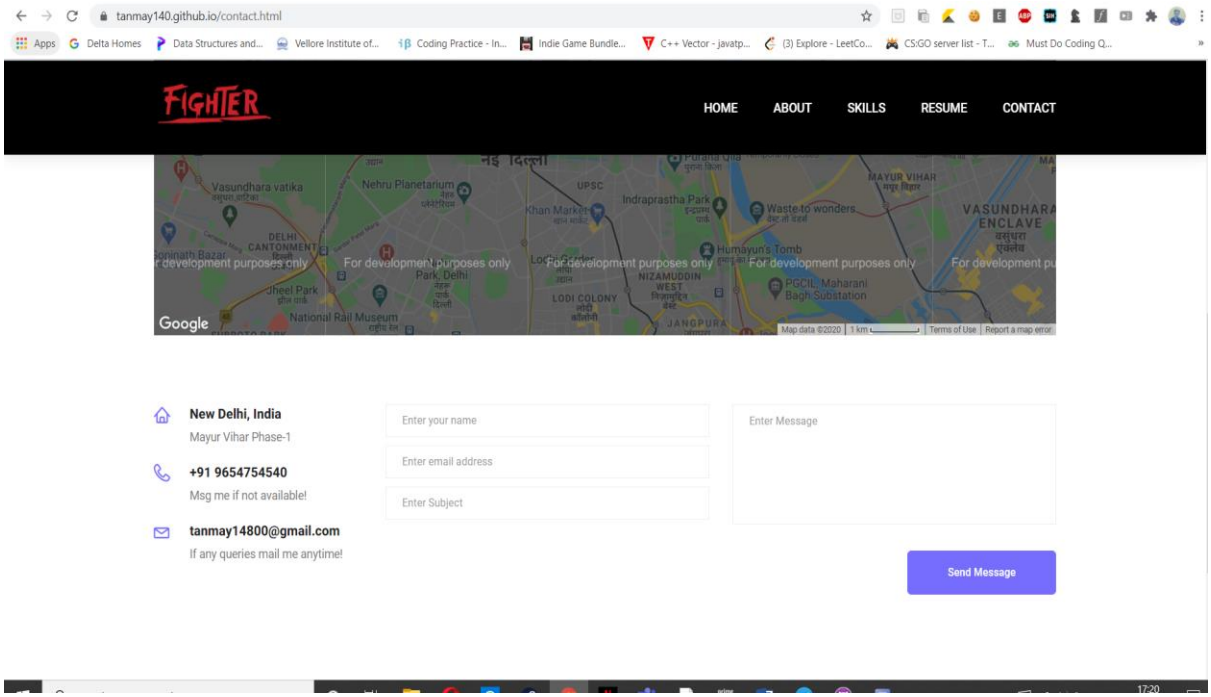
Skills:



Resume (file open on clicking):



Contact page:



- **Write down the pros and cons of GitHub.**

PROS:

- **Version Control:** GitHub, being built over Git, makes it fast and easy to develop projects in versions/branches and easily rollback to previous versions when necessary.
- **Pull Requests/Review:** GitHub has a powerful UI for creating pull requests, with useful tools like inline commenting and more recently "suggested changes". Pull request history is always maintained and easy to search.
- **Collaboration/Auditing:** It's easy for multiple team members to work on the same project and merge changes (often) seamlessly. All contributions are tracked so it's easy to identify contributors.
- **Industry Standard:** GitHub is used by virtually all major open source projects so it's very easy to find and contribute to projects of interest if you're well versed with GitHub.
- Bring the social aspect of programming into focus

CONS:

- Reviewing large pull requests can be tedious and it can be tough to identify recent changes (e.g. a single line change) in new files or files with lots of changes.
- It should be a bit harder to push unresolved merge conflicts, we've had these slip through once in a while.
- You have to be careful with merge operations; a bad merge can be painful to reverse.

- GitHub's status as an industry leader means it's often targeted by sophisticated attackers with DDOS attacks, which has kicked it offline a handful of times in the past few years.
- Conflict management could be improved.
- When browsing history of a file, GitHub could make it easier to see the file after a particular commit instead of just being able to quickly view the commit. I'd like to be able to see the commit or the file itself in one click.

- **<u>List down the features needs to be added in GitHub.</u>**
  - I think GitHub should incorporate two-factor authentication to improve user account security.
  - Free repositories have a size limit of 1GB which can be increased.
  - Users without paid accounts should have the access to make at least 2 private repositories.
  - Improved security scanning (for keys in history to prevent merges, etc).
  - UI - Although there is a readme file that can be made to look pretty, over the UI is very dry.

- No cross-repo issue tracking, hard to see all open pull requests at once.

- **<u>Compare the minimum of three version control applications.</u>**

3 version control systems are

Git ,

Apache Subversion (SVN),

and **CVS.**

- **<u>CVS</u>** has been around since the 80s, and has been very popular with both commercial and open source developers. It is released under the GNU license, and uses a system to let users “check out” the code they are going to work on and “check in” their changes.
- The **CVS server** runs on Unix-like systems with client software that runs on multiple operating systems. It is considered the most mature version control system because it has been developed for such a long time and does not receive many requests for new features at this time.

  Pros:

- Has been in use for many years and is considered mature technology

Cons:

- Moving or renaming files does not include a version update
- Security risks from symbolic links to files

- **SVN** was created as an alternative to CVS that would fix some bugs in the CVS system while maintaining high compatibility with it.
- Like CVS, SVN is free and open source with the difference of being distributed under the Apache license as opposed to GNU.
- To prevent corruption in the database from being corrupted, SVN employs a concept called atomic operations. Either all of the changes made to the source are applied or none are applied, meaning that no partial changes will break the original source.
- Many developers have switched to SVN as it is a newer technology that takes the best features of CVS and improves upon them. While CVS's branch operations are expensive and do not really lend themselves to long-term forks in the project, SVN is designed to allow for it, lending itself better to large, forked projects with many directions.

Pros:

- Newer system based on CVS
- Includes atomic operations
- Cheaper branch operations

Cons:

- Still contains bugs relating to renaming files and directories
- Insufficient repository management commands
- Slower comparative speed

Git

- First developed by Linus Tor valds of Linux fame, **Git** takes a radical approach that differs greatly from CVS and SVN.
- The original concepts for Git were to make a faster, distributed revision control system that would openly defy conventions and practices used in CVS. It is primarily developed for Linux and has the highest speeds on there.
- It will also run on other Unix-like systems, and native ports of Git are available for Windows as msysgit. As there is no centralized server, Git does not lend itself to single developer projects or small teams as the code may not necessarily be available when using a non-repository

computer. Workarounds exist for this problem, and some see Git's improved speed as a decent tradeoff for the hassle.

- Git also comes equipped with a wide variety of tools to help users navigate the history system. Each instance of the source contains the entire history tree, which can be useful when developing without an internet connection.

Pros:

- Great for those who hate CVS/SVN
- Dramatic increase in operation speed
- Cheap branch operations
- Full history tree available offline

Cons:

- Learning curve for those used to SVN
- Not optimal for single developers
- Limited Windows support compared to Linux

Conclusion:

For the most part, people use CVS because they are already used to it, and while some of the quirks and limitations are bothersome, they've already figured out how to make a work around the given limitations and have those workarounds implemented. Especially if your team is currently engaged on a certain project, the prospect of migrating everything to another revision control is annoying, and if they were to switch, it would most likely be to SVN.

Git has a clear speed improvement over its competitors, and for projects that lend themselves to distributed systems, it is a clear improvement.

The primary downside cited for Git is that it can be at times difficult to explain to others, and there is likely to be a slow down in production as programmers adapt to it. Once it is learned, however, the speed increases and better branch management will reclaim that time and more.